Name: Nikhil Kumar

Roll: 1806055 , CSE 6th Sem

Information Security Assignment 01

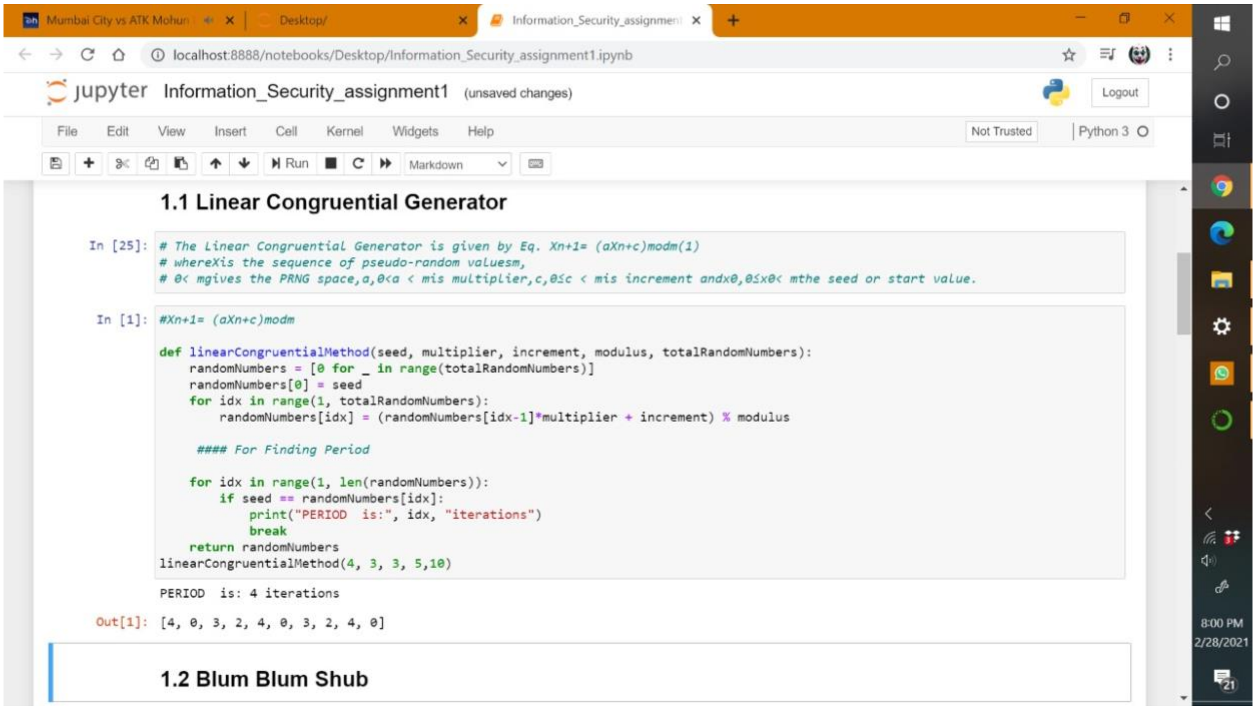
Q1: (a and b)

Linear Congruential Generator

code:

```
#Xn+1= (aXn+c)modm

def. linearCongruentialMethod(seed, multiplier, increment, modulus, totalRandomNumbers):

 randomNumbers = [0 for _ in range(totalRandomNumbers)]

 randomNumbers[0] = seed

 for idx in range(1, totalRandomNumbers):

 randomNumbers[idx] = (randomNumbers[idx-1]*multiplier + increment) % modulus

 #### For Finding Period

 for idx in range(1, len(randomNumbers)):

 if seed == randomNumbers[idx]:

 print("period is:", idx, "iterations")

 break

 return randomNumbers
```

linearCongruentialMethod(4, 3, 3, 5,10)



output:

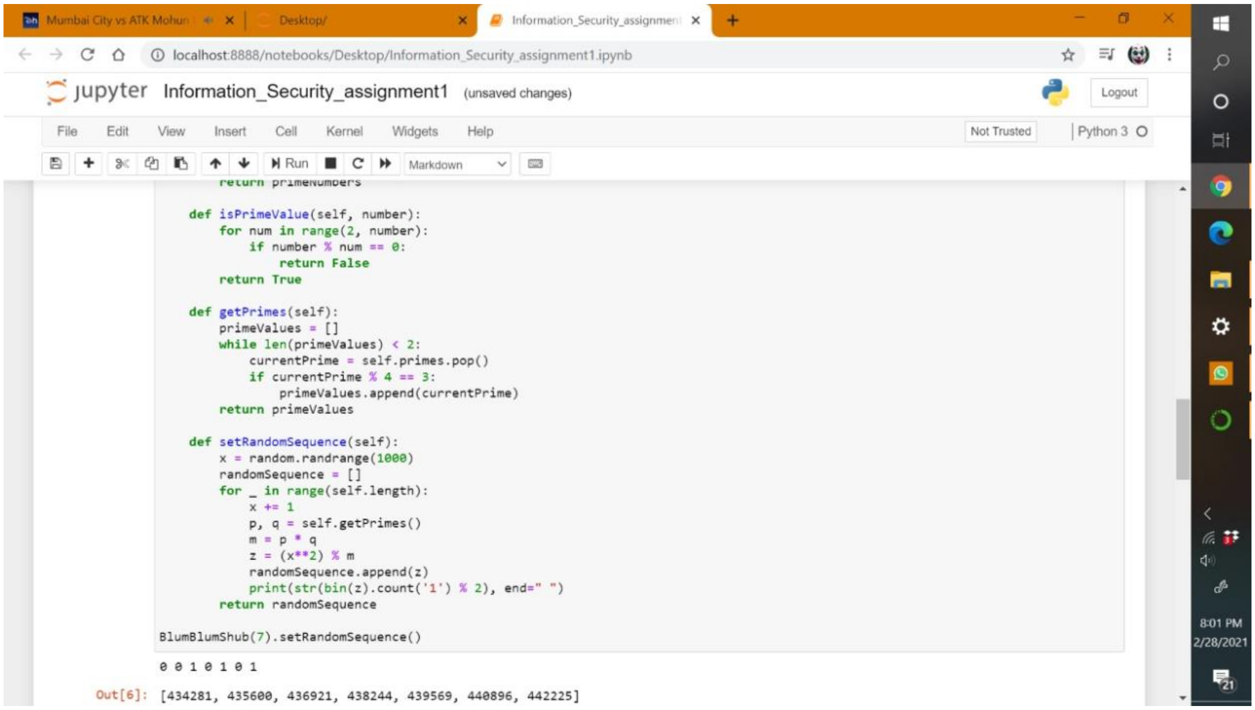
Blum Blum Shub

Code:

```
##xn+1=x2nmodM
import random
class BlumBlumShub:
 def __init__(self, length):
 self.length = length
 self.primes = self.generatePrimes(1000)

 def generatePrimes(self, number):
 primeNumbers = []
 for num in range(number):
 if self.isPrimeValue(num):
 primeNumbers.append(num)
```

```
return primeNumbers

def isPrimeValue(self, number):
for num in range(2, number):
if number % num == 0:
return False
return True

def getPrimes(self):
primeValues = []
while len(primeValues) < 2:
currentPrime = self.primes.pop()
if currentPrime % 4 == 3:
primeValues.append(currentPrime)
return primeValues

def setRandomSequence(self):
x = random.randrange(1000)
randomSequence = []
for _ in range(self.length):
x += 1
p, q = self.getPrimes()
m = p * q
z = (x**2) % m
randomSequence.append(z)
print(str(bin(z).count('1') % 2), end=" ")
return randomSequence
BlumBlumShub(7).setRandomSequence()
```

output:



Linear Feedback Shift Register

Code:

```
def linearFeedBackShiftRegister(seed, postionOfTaps):
 shiftRegister, xor = seed, 0
 period = 0
 while True:
 for position in postionOfTaps:
 period += 1
 A = int(shiftRegister[len(shiftRegister)-position])
 B = int(shiftRegister[0])
 notA = 0 if A == 1 else 1
 notB = 0 if B == 1 else 1
 xor = A*notB + notA*B
 shiftRegister= shiftRegister[1:] + str(xor)
 print((shiftRegister, xor), end=" ")
```

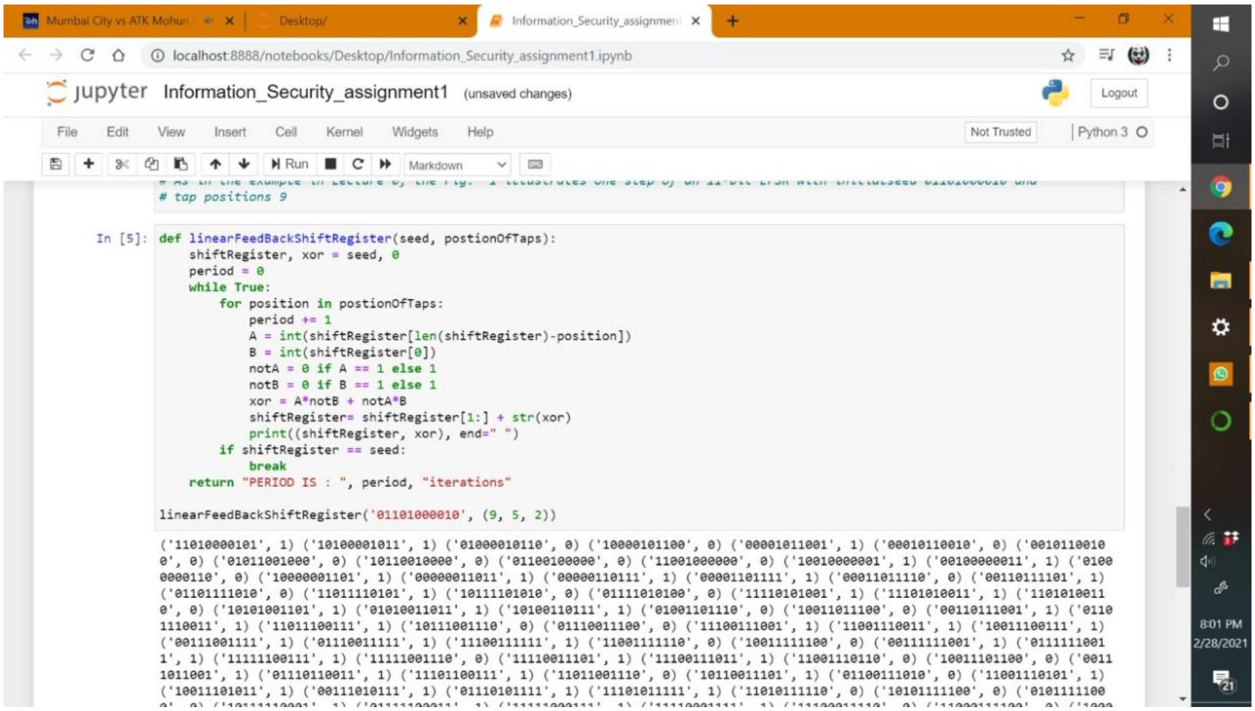
if shiftRegister == seed:

break

return "period is: ", period, "iterations"

linearFeedBackShiftRegister('01101000010', (9, 5, 2))

output:



Q2: Solution

A random number generator needs to be secure against attack by an adversary who knows the algorithm and a (large) number of previously generated bits in order to be considered cryptographically secure. What this means is that someone with that information can't reconstruct any of the hidden internal state of the generator and give predictions of what the next bits produced will be with better than 50% accuracy. Normal pseudo-random number generators are generally not cryptographically secure, as reconstructing the internal state from previously output bits is generaly trivial (often, the entire internal state is just the last N bits produced directly). Any random number generator without good statistical properties is also not cryptographically secure, as its output is at least party predictable even without knowing the internal state.

Any good crypto system can be used as a cryptographically secure random number generator use the crypto system to encrypt the output of a 'normal' random number generator. Since an adversary can't reconstruct the plaintext output of the normal random number generator, he can't attack it directly. This is a somewhat circular definition an begs the question of how you key the crypto system to keep it secure, which is a whole other problem.

Name: Nikhil kumar

Roll no:1806055